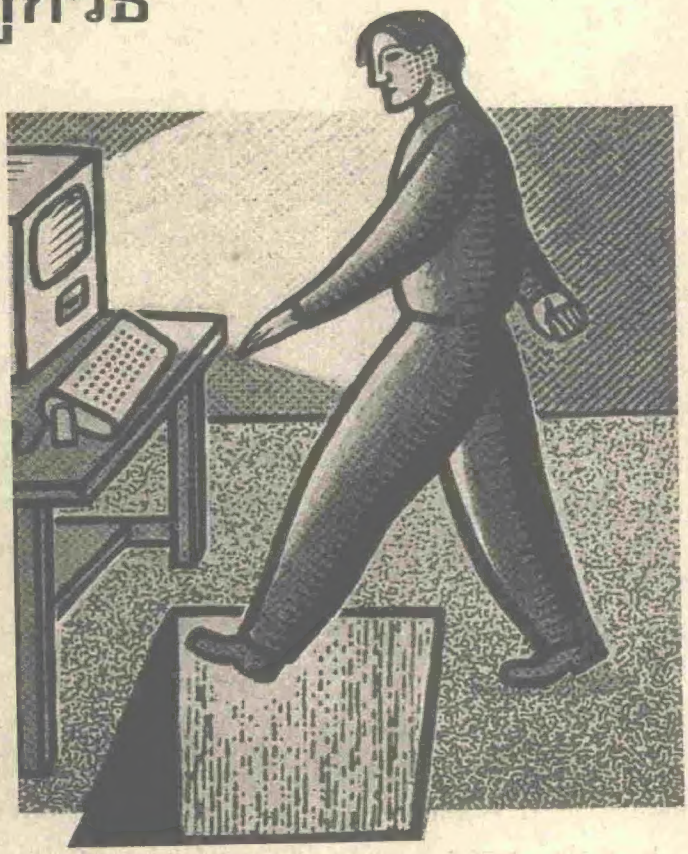
อนันต์ กิตติบรรดุ

มาเรียนรู้ การสร้างซอฟต์แวร์ให้ถูกวิธี



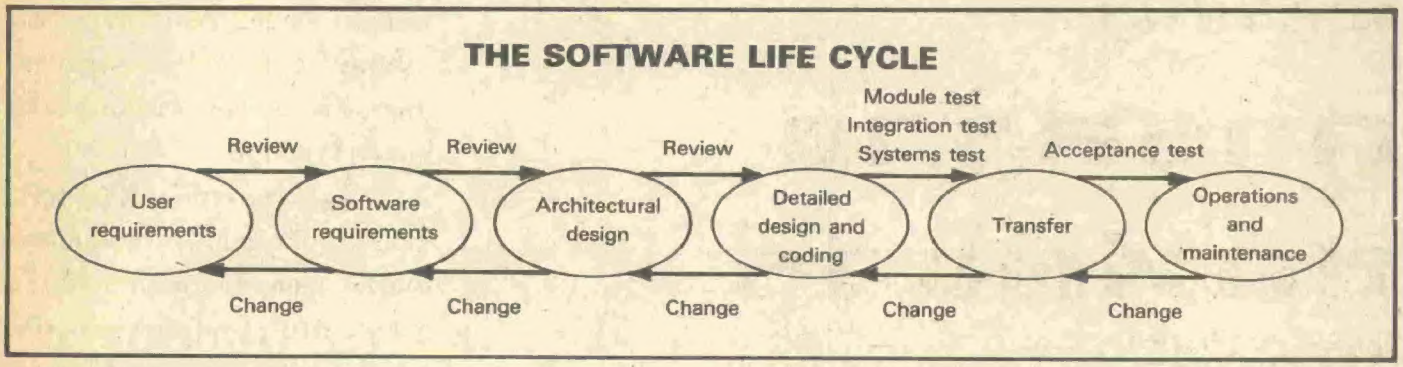
เพิ่มทักษะในการเรียนรู้ เข้าใจหลักวิศวกรรมซอฟต์แวร์กันอย่างคร่าว ๆ เรามีคำตอบให้คุณที่นี้

จากประสบการณ์ที่ผ่านมาก็หลายปีพอสมควร ในการพัฒนาซอฟต์แวร์มาก็หลายตัว ทั้งพัฒนาขึ้นมาเพื่อใช้งานเอง หรือถูกบังคับให้ต้องทำ (ถ้าไม่ทำ...ก็ไม่จบ!) หรือแม้แต่ทำงานร่วมกันเป็นกลุ่มในงานขนาดใหญ่ ๆ ใช้ระยะเวลานาน มีค่าใช้จ่ายสูง เช่น งานวางระบบ ซึ่งงานที่ผ่านมาก็จัดว่ามากพอสมควร แต่ผมเองกลับมีความรู้สึกว่า การพัฒนาซอฟต์แวร์ที่แล้วมานั้นยังทำแบบสเปะสปะ ไม่ค่อยเป็นที่พอใจนัก และก็มักจะมาคิดว่า "งานหน้าต้องทำให้ดีกว่านี้" เพราะมีประสบการณ์มาแล้ว แต่ลงท้ายก็เหมือนเดิม คือผิดหวัง ครับผม

เนื่องด้วยการพัฒนาซอฟต์แวร์ขึ้นมาสักตัวนั้น เป็นเรื่องที่ยากพอสมควร ถ้าเปรียบไปแล้วน่าจะพอกับการปีนขึ้นภูเขาสูง ทำไมหรือครับก็เพราะการปีนภูเขานั้นจะต้องใช้น้ำอดและน้ำทนสูงมาก ต้องทนกับบรรยากาศที่คอยกดดันอยู่ตลอดเวลา และหากเกิดการผิดพลาดหรือพลาดพลั้งขึ้นเมื่อใด แม้จะเป็นเพียงเล็กน้อย แต่ความเสียหายนั้นอาจถึงชีวิตกันเลยทีเดียว

ทีนี้ถ้าจะมองกันในเรื่องของความผิดพลาดที่ว่าอาจจะสาหัสสากรรจ์ดังกล่าวแล้วนั้น มันก็มาจากสาเหตุหลายประการด้วยกัน อาทิเช่น ขาดประสบการณ์ ขาดทักษะหรือความเชี่ยวชาญ ความเร่งรีบจนเกินไป การสำรวจหรือเซอร์เวย์งานไม่ครบถ้วน การคาดการณ์งานที่ไม่ถูกต้อง หรือแม้แต่การทดสอบข้อผิดพลาดที่ทำไม่ทั่วถึง เหล่านี้เป็นต้น ล้วนแต่ทำให้งานพัฒนาซอฟต์แวร์เหล่านั้นออกมาไม่ดีพอ ทำงานได้ก็พอ (just worked) บ้าง หรือเป็นแบบเร็วแต่ใช้ไม่ได้ (quick and dirty program) บ้าง แล้วก็มักจะมีคำถามเหล่านี้ตามมาเสมอว่า

- ทำไมเราไม่พบข้อผิดพลาด แต่เมื่อนำซอฟต์แวร์ไปใช้งานจริงแล้วกลับมาพบข้อผิดพลาด
- ทำไมโปรแกรมต่าง ๆ ที่ทำกันจึงใช้ระยะเวลานานมากกว่าจะเสร็จสิ้น
- ทำไมถึงสิ้นเปลืองค่าใช้จ่ายสูงมาก (งบบานปลาย)
- ทำไมการที่จะทราบถึงความก้าวหน้าของการพัฒนางานที่ทำอยู่จึงยากหรือผิดคาดเหลือเกิน



รูปที่ 1 วงจรพัฒนาแบบวอเตอร์ฟอลล์

จากปัญหาคร่าว ๆ ที่ผมนำมายกตัวอย่างให้ดูกันนี้แหละครับ จึงได้เกิดหลักวิชาวิศวกรรมซอฟต์แวร์ หรือ Software Engineering ขึ้น ซึ่งคอลัมน์ของผมฉบับนี้จึงใคร่ขอนำท่านผู้อ่านผู้มีส่วนต่อการพัฒนาซอฟต์แวร์ ได้มาทำความรู้จักหรือให้ความสนใจกับเรื่องของวิศวกรรมซอฟต์แวร์นี้ให้มากขึ้น ซึ่งก็คงจะทำให้ปัญหาหรือคำถามต่าง ๆ เหล่านี้ลดน้อยลงในการพัฒนาซอฟต์แวร์ครั้งต่อไป

องค์ประกอบของวิศวกรรมซอฟต์แวร์

วิศวกรรมซอฟต์แวร์นั้นโดยหลักคร่าว ๆ แล้วสามารถแยกออกได้ 3 หัวข้อด้วยกัน ได้แก่

- วิธีการ (Methods)
- ทูลส์ (Tools)
- โพรซีเดอร์ (Procedures)

วิธีการ (Methods)

วิธีการ ส่วนนี้จะมองในแง่เทคนิคคัลว่าจะสร้างซอฟต์แวร์ได้อย่างไร (how to's) โดยมุ่งไปที่ประเด็นเหล่านี้

- การวางแผนและการประเมินงาน (Project planning and esti mation)
- การวิเคราะห์หาความต้องการของระบบ (System requirements analysis)
- การออกแบบโครงสร้างข้อมูล (Data structure) สถาปัตยกรรมของโปรแกรม (Program architecture) และอัลกอริทึม
- การสร้างเป็นภาษาโปรแกรม (Coding)
- การทดสอบ (Testing)
- การดูแลรักษา (Maintenance)

ทูลส์ (Tools)

เป็นเครื่องมือที่จะต้องจัดหามาใช้เพื่อซัพพอร์ต หรือสนับสนุนให้วิธีการสำเร็จได้ด้วยดีและรวดเร็ว สำหรับเครื่องมือของวิศวกรรมซอฟต์แวร์ที่เป็นที่รู้จักกันดีก็คือ CASE (Computer Aided Software Engineering) ซึ่งจะช่วยทำให้ภารวิเคราะห์, ออกแบบ, การโค้ด และการทดสอบโปรแกรมตามขั้นตอนของวิธีการเป็นไปได้อย่างมีประสิทธิภาพ

โพรซีเดอร์ (Procedure)

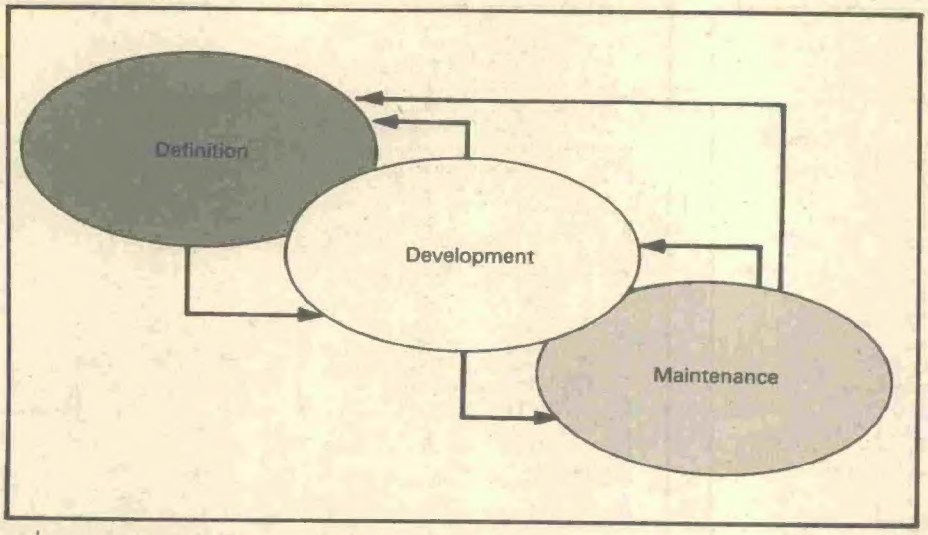
เป็นขั้นตอนของงานที่จะรวมเอาวิธีการกับทูลส์เข้าไว้ด้วยกัน และจัดแบ่งระยะเวลาของการพัฒนางานให้เหมาะสม พร้อมกับคอยคุมงานให้เป็นไปตามเป้าหมาย ถูกต้อง และมีคุณภาพ

จากองค์ประกอบของซอฟต์แวร์ทั้ง 3 นั้น คราวนี้จะมาดูถึงวงจรพัฒนาระบบ (System Development life cycle) ที่นิยมใช้กัน

Waterfall life cycle

ในบางครั้งก็เรียกกันว่า Waterfall model เป็นโมเดลที่อาจกล่าวได้ว่าเป็นแบบคลาสสิคเป็นตัวแรกที่เป็นที่ยอมรับกัน และในปัจจุบันเองก็ยังมีใช้กันอยู่บ้าง จัดว่าเป็นแม่แบบให้โมเดลอื่น ๆ นำมาใช้กันด้วย โมเดลนี้กำเนิดขึ้นในปี ค.ศ. 1970 โดยนายรอยซ์ (Royce's waterfall) ซึ่งในสมัยนั้นซอฟต์แวร์เป็นงานที่หนักหนามากต้องเสียทั้งเวลาและงบประมาณมหาศาล นายรอยซ์จึงได้มีความคิดที่จะแบ่งงานออกเป็นหลาย ๆ เฟสงาน ดังรูปที่ 1 โดยแบ่งไปตาม

- ความต้องการ (requirements)
- การออกแบบในระดับบนสุด (top-level design) ซึ่งจะเป็นการรวบรวมความต้องการทั้งหมดของระบบ แล้วนำมาวิเคราะห์ว่าจะมีอะไรบ้างที่จะมาทำเป็นซอฟต์แวร์
- การออกแบบในระดับรายละเอียด (detailed design) เป็นการมองลึกลงไปถึงโครงสร้างข้อมูล (data structure) อัลกอริทึมที่ใช้ฟังก์ชันหรือโพรซีเดอร์



รูปที่ 2 มุมมองร่วมกันของวิศวกรรมซอฟต์แวร์

- การโยกย้าย (transfer) เป็นการตรวจสอบว่าการทำงานถูกต้องทั้งหมดหรือไม่ และตรงตามความต้องการของผู้ใช้หรือไม่ ก่อนที่จะส่งไปถึงมือผู้ใช้เพื่อไปใช้งานจริงต่อไป

- การดูแลรักษา (maintenance) เป็นการดูแลไม่ให้ซอฟต์แวร์นั้นผิดพลาด หากผิดพลาดก็ทำการแก้ไข การปรับปรุงให้ทำงานได้กับสภาพแวดล้อมใหม่ ๆ รวมถึงการเพิ่มเติมการทำงาน

ปัญหาของวอเตอร์ฟอลล์ คือ มันไม่เหมาะกับงานโปรเจ็กต์ที่มีขนาดใหญ่และซับซ้อน เพราะเป็นการยากที่จะทำให้การโฟลว์ หรือไหลของแต่ละเฟสงานเป็นระบบระเบียบ และเมื่อมีปัญหาเกิดขึ้นแต่ละเฟสจะทำได้ช้าไม่สะดวก โดยเฉพาะอย่างยิ่งในเฟสของความต้องการของยูสเซอร์จะต้องกลับไปเริ่มที่เฟสแรกใหม่ ทำให้เสียทั้งเวลาและทรัพยากรอีกมาก ส่วนอีกปัญหาก็คือ ผู้ใช้ต้องรอระบบงานเป็นเวลานานมาก เพราะกว่าจะนำไปใช้ได้จริง ๆ ก็ต่อเมื่อระบบงานสั้นสุดจนเฟสสุดท้ายแล้ว

โปรโตไทพ์ (Prototype)

โปรโตไทพ์ หรือต้นแบบ ไม่ใช่เป็นวงจรพัฒนาระบบงานหรือไลฟ์ไซเคิล แต่เป็นเทคนิคที่มักนำมาใช้เพิ่มเติมเพื่อแก้ปัญหา ในส่วนของการเซอร์เวย์ความต้องการของยูสเซอร์ อันเป็นส่วนหัวใจของระบบงาน ซึ่งยูสเซอร์อาจไม่สามารถบอกความต้องการได้ละเอียดเพียงพอ กอปรกับต้นแบบจะช่วยให้ระบบงานที่จะพัฒนาขึ้นมาจริง ๆ ภายหลัง เป็นไปตามความต้องการของยูสเซอร์มากที่สุด เพราะยูสเซอร์นั้นจะบอกแต่เพียงจุดประสงค์ทั่ว ๆ ไป ซึ่งเป็นข้อมูลด้านนามธรรม (Realism) ไม่ได้ให้ข้อมูลเกี่ยวกับรายละเอียดของข้อมูลเข้า ข้อมูลออก หรือโปรเซสของงาน ซึ่งเป็นรูปแบบด้านรูปธรรม (abstract solution) ที่ผู้พัฒนาจะต้องมาจัดการเองก่อนที่จะนำมาพัฒนากับคอมพิวเตอร์ และอีกส่วนหนึ่งทำให้ผู้พัฒนามีความมั่นใจเพิ่มขึ้นว่างานที่ได้จะมีประสิทธิภาพ มีอัลกอริทึมที่ถูกต้อง และเป็นที่ถูกใจของยูสเซอร์อย่างแท้จริง

มุมมองร่วมกันของวิศวกรรมซอฟต์แวร์

จากวงจรพัฒนาระบบงานหรือไลฟ์ไซเคิล แบบวอเตอร์ฟอลล์ หรืออื่น ๆ ก็ตาม ถ้าจะว่าไปแล้วก็จะมีมุมมองร่วมกันอยู่เป็น 3 เฟสงานด้วยกัน ดังรูปที่ 2 ได้แก่

1. ช่วงหาข้อกำหนด (Definition phases)

2. ช่วงทำการพัฒนา (Development phases)

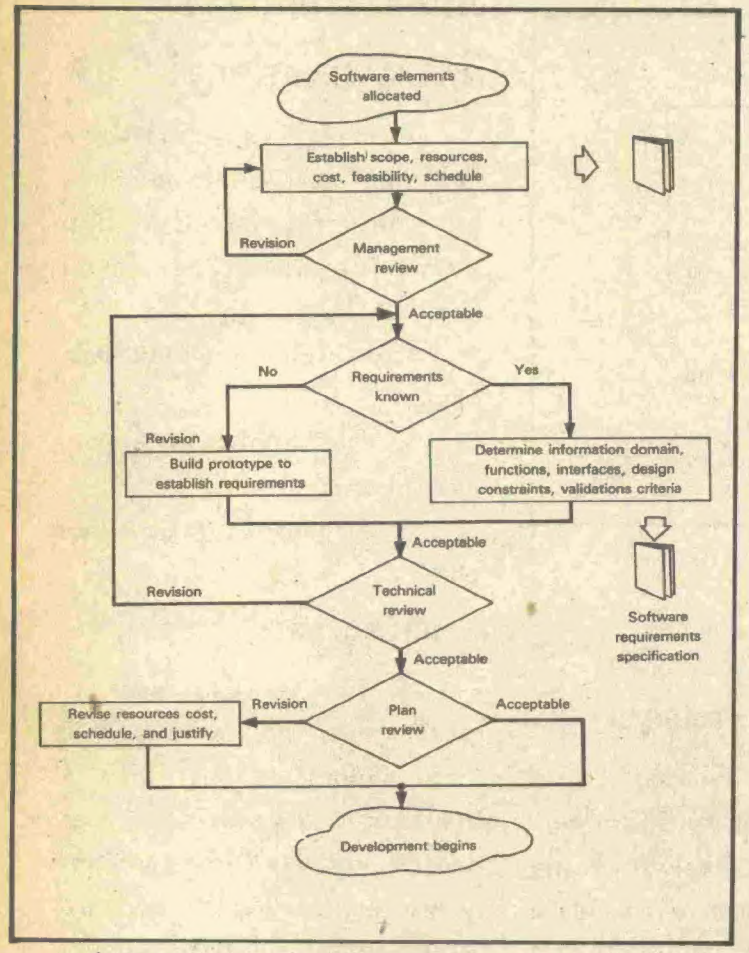
3. ช่วงดูแลรักษา (Maintenance phases)

ช่วงหาข้อกำหนด

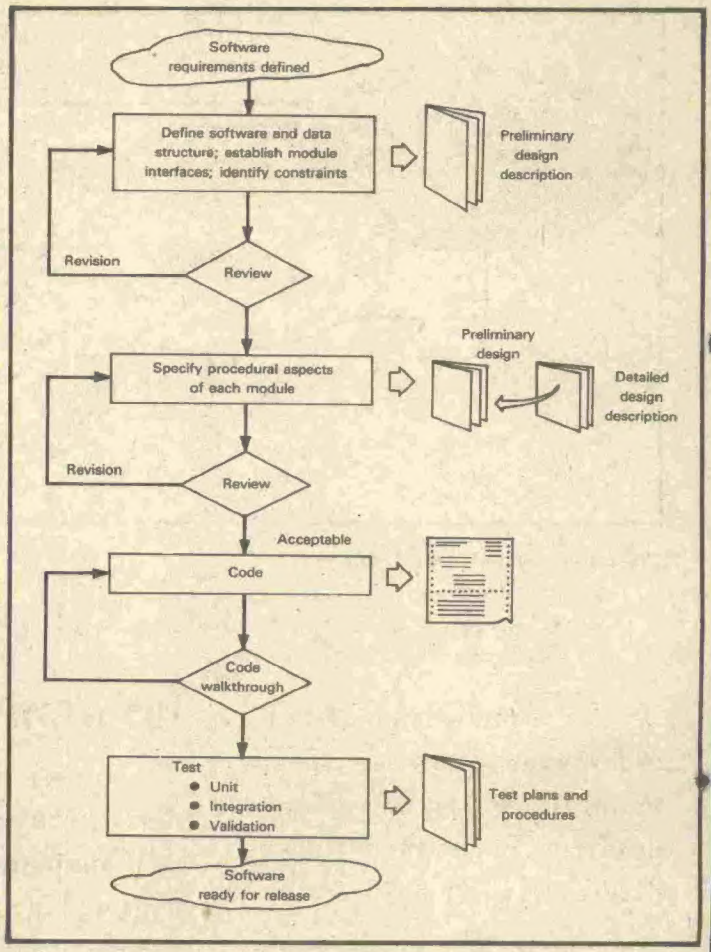
ช่วงนี้จะมุ่งที่ประเด็นคำถาม "อะไร (what)" เช่น จะหาข้อมูลสำคัญ (information) อะไรมาโปรเซส มีฟังก์ชันอะไร จะต้องต่อเชื่อม (interface) กับอะไรบ้าง ข้อกำหนดขอบเขต (constraints) ของระบบมีอะไรบ้าง เป็นต้น นั่นก็คือ ช่วงหาข้อกำหนดจะมีด้วยกัน 3 ขั้นตอน

- การวิเคราะห์ระบบ (System Analysis) ขั้นตอนนี้นักวิเคราะห์ระบบและวิศวกรระบบจะมีหน้าที่ในการเก็บรวบรวมความต้องการ (requirement) ของระบบทั้งหมด แล้วนำมาวิเคราะห์ว่าจะนำส่วนใดบ้างมาทำเป็นซอฟต์แวร์ และจะมองไปถึงเรื่องของฮาร์ดแวร์ที่จะใช้ บุคคลที่เกี่ยวข้องกับระบบหรือพวกระบบฐานข้อมูลที่เหมาะสมด้วย

- การวางแผนซอฟต์แวร์ (Software Project Analysis) ขั้นตอนนี้จะหาขอบเขต (scope) ของซอฟต์แวร์ การจัดหาทรัพยากร (Resource) การประเมินราคา รวมถึงการกำหนดระยะเวลางาน โดยทำเป็นตารางเวลา (Schedules)



รูปที่ 3 ผังงานช่วงการกำหนดความต้องการของระบบ



รูปที่ 4 ผังงานช่วงการพัฒนา

- การวิเคราะห์ความต้องการ (Requirement Analysis) ขั้นตอนนี้จะวิเคราะห์ออกมาว่าเป็นไปตามความต้องการของยูสเซอร์หรือยัง ถ้ายังต้องมีการแก้ไขหรืออาจจำเป็นต้องสร้างต้นแบบหรือโปรโตไทพ์ (Prototype) ขึ้นมา เพื่อที่จะนำเอารีไควร์เมนต์เหล่านี้มาเขียนเป็น "ข้อกำหนดความต้องการของระบบ (Software requirement specification)" ซึ่งจะมีรายละเอียดว่ามีฟังก์ชันอะไร จะต้องต่อเชื่อมกับอะไร ฯลฯ ดังที่เสนอไปแล้ว

ช่วงข้อกำหนดนี้จะเขียนเป็นผังงานได้ดังรูปที่ 3

ช่วงทำการพัฒนา

ช่วงนี้จะมุ่งที่ประเด็นคำถาม "อย่างไร (How)" ในระหว่างที่ทำการพัฒนาระบบงานอยู่ เช่น จะออกแบบโครงสร้างข้อมูล และสถาปัตยกรรมของซอฟต์แวร์ (Software architecture) อย่างไร จะให้รายละเอียดในแต่ละโมดูล (Module) เป็นอย่างไรก่อนที่จะทำการโค้ดดิ้ง (coding)

ซึ่งหลังจากเสร็จสิ้นจุดนี้จะได้ออกมาเป็น "รายละเอียดของการออกแบบ (Detailed design)" แล้วจึงนำเอามาออกแบบกันอีกทีว่าจะทำให้เป็นรูปของโปรแกรมมิงแลงเกจได้อย่างไร และท้ายสุดก็มาคิดว่าจะทำการทดสอบระบบได้อย่างไร

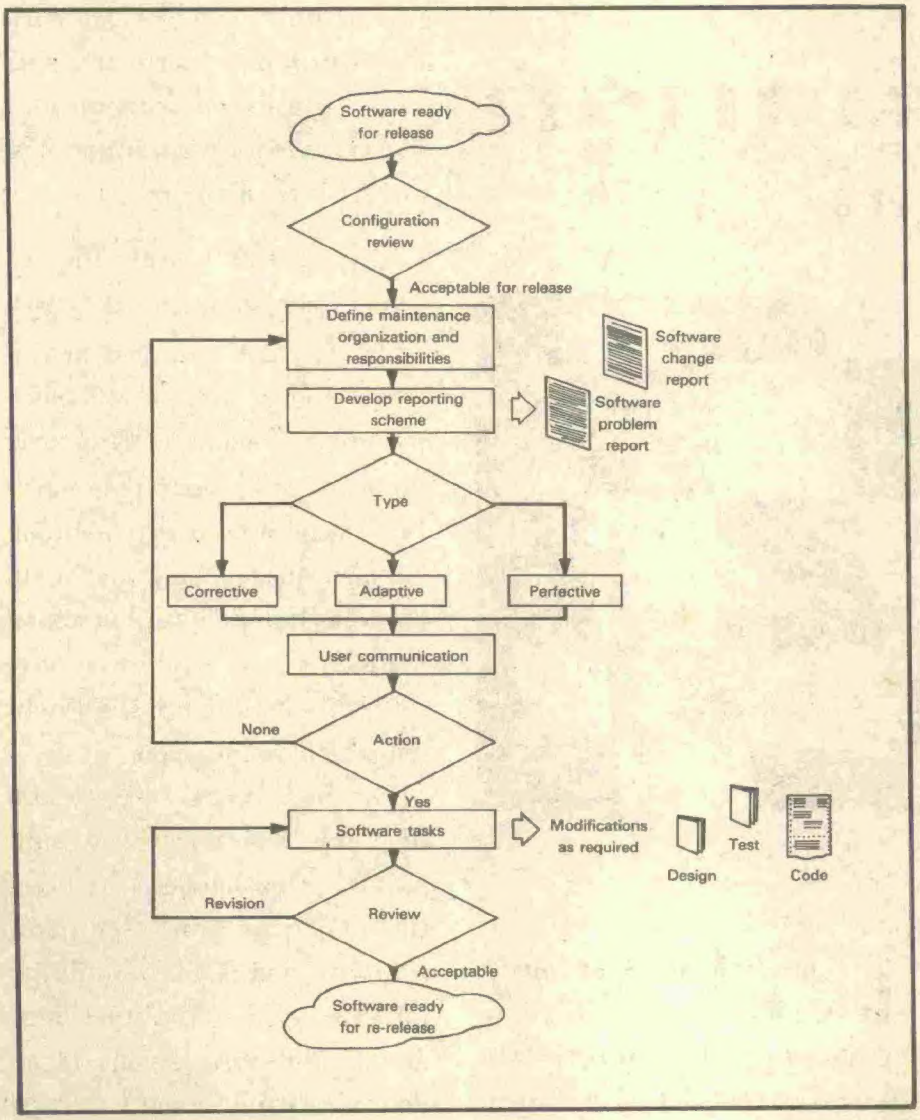
สรุปก็คือ ช่วงทำการพัฒนาจะมีอยู่ 3 ขั้นตอน คือ

- ออกแบบซอฟต์แวร์ (Software design)

- การสร้างเป็นภาษาโปรแกรม (Coding)

- การทดสอบระบบ (Testing)

ช่วงทำการพัฒนานี้จะเขียนเป็นผังงานได้ดังรูปที่ 4



รูปที่ 5 ผังงานช่วงดูแลรักษา

ช่วงดูแลรักษา

ช่วงนี้จะมุ่งไปที่ประเด็น "การเปลี่ยนแปลง (change)" ที่จะต้องมีขึ้นไม่ว่าจะเป็น

- การแก้ไขข้อผิดพลาด (Error Correction) ซึ่งจะต้องมีอยู่อย่างแน่นอน แต่จะมากน้อยเท่านั้น ข้อผิดพลาดเหล่านี้ยูสเซอร์จะเป็นผู้ค้นพบหลังจากนำระบบไปใช้งานแล้ว ขั้นตอนนี้เรียกว่า "Corrective maintename"

- การปรับปรุง (Adaptation) เพื่อทำให้ระบบสามารถที่จะใช้งานต่อไปตามสภาพแวดล้อมใหม่ ๆ เทคโนโลยีใหม่ ๆ ที่เข้ามา เช่น ระบบปฏิบัติการ, ซีพียู อุปกรณ์เสริมภายนอก (peripheral) เป็นต้น ขั้นตอนนี้เรียกว่า "Adaptive maintenance"

- การปรับเปลี่ยนให้มีประสิทธิภาพขึ้น (Enhancement) เช่นยูสเซอร์อาจต้องการให้การทำงานเร็วขึ้น ต้องการเพิ่มเติมการทำงาน โดยขั้นตอนนี้เรียกว่า "Perfective maintenance"

ช่วงนี้สามารถแสดงเป็นผังงานได้ดังรูปที่ 5

ส่งท้าย

เป็นอย่างไรบ้างครับท่านผู้อ่าน ผมหวังว่าข้อเขียนนี้คงทำให้ท่านผู้อ่านรู้และเข้าใจหลักวิศวกรรมซอฟต์แวร์กันคร่าว ๆ แล้วนะครับ ถ้าหากท่านผู้อ่านสนใจอยากหาความรู้เพิ่มเติม ก็อาจจะหาอ่านได้จากหนังสือซอฟต์แวร์เอ็นยิเนียริ่งที่มีวางขายตามร้านหนังสือนะครับ ซึ่งบทความนี้คงทำให้ท่านผู้อ่านได้เข้าใจหลักการของวิชานี้กันเท่านั้น เพราะถ้าจะว่ากันให้ละเอียดลึกซึ้งนั้นอาจถึงกับต้องศึกษากันถึงระดับปริญญาโทหรือเอกทีเดียวครับ สวัสดี

หนังสืออ้างอิง

Roger S.Pressman, "Software Engineering : A Practitioner's Approach", McGRAW-HILL, 1987